SYSTEM AND METHOD FOR VALIDATING HIERARCHICALLY-ORGANIZED MESSAGES

FIELD OF THE INVENTION

[0001] The present invention relates generally to the field of computing, and, more particularly, to systems and methods for message validation.

BACKGROUND OF THE INVENTION

[0002] Information is generally represented in a lexicon or language that is sufficiently rich to allow both valid and invalid content to be expressed. For example, it is possible to use the Roman alphabet to write a correct English sentence, but it is also possible to string together English words that do not obey the semantic or syntactic rules of any language, or to string together English letters in a manner that is completely unintelligible. The languages in which computer data is expressed are no exception – i.e., it is possible to write computer data that is not valid according to some set of rules.

[0003] In computer systems, much data is expressed in a hierarchical manner, such as in the form of an eXtensible Markup Language (XML) message. An XML message conforms to some schema, which essentially defines the proper syntax of some class of messages. For example, a type of message may be an "address," and the schema for an address may require that an address include

a street name, a city, a state, and a zip code. However, even a message that obeys the schema may be invalid for some substantive reason. For example, any combination of data that purports to be a street name, city, state, and zip code would satisfy the schema, but the address may still be invalid if, say, the state element is not the name of one of the United States, or if the zip code specified does not match the city/state combination.

[0004] The traditional way to do the validation is through brute force, message-specific code. The validation procedure for each message class would have to be written separately with no way in which to modify the procedure's behavior without modifying the class code itself. The problem with this technique is that any change to the substantive requirements for validation would require a change to the source code. Not only is such a change to the source code cumbersome, but it also may require in some cases that the source code be distributed to the public, so that a consumer of the validation procedure can make custom modifications to the procedure. Such distribution of source code may be undesirable.

[0005] In view of the foregoing, there is a need for a system that overcomes the drawbacks of the prior art.

SUMMARY OF THE INVENTION

[0006] The present invention provides a mechanism for validating a message. A validation engine walks through the message. For example, in the case of an XML message which can be viewed as a tree data structure, the validation engine may walk through the nodes of the tree in a depth-first traversal order.

[0007] When each node is encountered, a table is consulted, which specifies "delegates" to validate the node. A delegate is a named unit of code that can be invoked by the validation engine. The validation engine invokes the delegate that corresponds to the type of node that has been encountered. The validation engine then traverses the subtrees of the current node. As these subtrees are traversed, the tables may be consulted and delegates are invoked in the manner described above. After the subtrees have been traversed, the validation engine consults the table again to determine whether there is a post-handler delegate for the current node. If such a post-handler is specified, then the post-handler is invoked. The delegates evaluate a node under some defined validation

standard (e.g., a delegate that validates a city, state, and zip code combination by determining that the combination actually corresponds to a real address in the United States).

[0008] Preferably, there is both a global table and a local table. The local table is consulted first to determine whether there is a delegate for the current node. If there is such a delegate, then that delegate is invoked. If no such delegate is named in the local table, then the global table is consulted to determine whether a delegate is named therein. If such a delegate is named in the global table, then the delegate from the global table is invoked. The global table may contain delegates that apply to a broad category of message classes, while the local table may contain delegates that are specific to a particular message class.

[0009] Preferably, the tables allow a delegate to be optionally specified as "exclusive." If a delegate is exclusive, then it has exclusive control over validating a node and its subtrees, so the subtrees are not traversed by the validation engine, and the delegates associated with the nodes in the subtrees are not applied by the validation engine. Otherwise, if a delegate is not exclusive, then the subtrees are traversed, and the delegates corresponding to the nodes in the subtrees are applied by the validation engine.

[0010] Other features of the invention are described below.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] The foregoing summary, as well as the following detailed description of preferred embodiments, is better understood when read in conjunction with the appended drawings. For the purpose of illustrating the invention, there is shown in the drawings example constructions of the invention; however, the invention is not limited to the specific methods and instrumentalities disclosed. In the drawings:

[0012] FIG. 1 is a block diagram of an example computing environment in which aspects of the invention may be implemented;

[0013] FIG. 2 is a block diagram of example message in the form of eXtensible Markup Language (XML);

[0014] FIG. 3 is a block diagram of a tree representation of the XML message of FIG. 2;

[0015] FIG. 4 is a block diagram of a tree, showing a depth-first traversal of the tree;

[0016] FIG. 5 is a block diagram of an example validation table in accordance with aspects of the invention; and

[0017] FIG. 6 is a flow diagram of an example validation process in accordance with aspects of the invention.

DETAILED DESCRIPTION OF THE INVENTION

Overview

[0018] Data that is used by a computer is typically organized in a hierarchical fashion – e.g., XML data that conforms to some schema. Even if data conforms syntactically to a schema, the data may still be invalid under some set of substantive rules. Traditionally, performing substantive validation of a collection of data (such as a message) requires special purpose code that is designed to validate a particular class of messages. The invention provides a general purpose tool that may be used to validate arbitrary classes of messages, and may be extended to apply an arbitrary set of rules to a message. The mechanisms of the present invention can be used to validate any arbitrary data or message for which the validation parameters and data can be expressed in XML form.

Exemplary Computing Environment

[0019] FIG. 1 shows an exemplary computing environment in which aspects of the invention may be implemented. The computing system environment 100 is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither should the computing environment 100 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary operating environment 100.

[0020] The invention is operational with numerous other general purpose or special purpose computing system environments or configurations. Examples of well known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but are not limited to, personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputers, mainframe computers, embedded systems, distributed computing environments that include any of the above systems or devices, and the like.

[0021] The invention may be described in the general context of computer-executable instructions, such as program modules, being executed by a computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network or other data transmission medium. In a distributed computing environment, program modules and other data may be located in both local and remote computer storage media including memory storage devices.

[0022] With reference to FIG. 1, an exemplary system for implementing the invention includes a general purpose computing device in the form of a computer 110. Components of computer 110 may include, but are not limited to, a processing unit 120, a system memory 130, and a system bus 121 that couples various system components including the system memory to the processing unit 120. The processing unit 120 may represent multiple logical processing units such as those supported on a multi-threaded processor. The system bus 121 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus (also known as Mezzanine bus). The system bus 121 may also be implemented as a point-to-point connection, switching fabric, or the like, among the communicating devices.

[0023] Computer 110 typically includes a variety of computer readable media. Computer readable media can be any available media that can be accessed by computer 110 and includes both volatile and nonvolatile media, removable and non-removable media. By way of example, and not limitation, computer readable media may comprise computer storage media and communication media. Computer storage media includes both volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CDROM, digital versatile disks (DVD) or other optical disk storage, magnetic

cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can accessed by computer 110. Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of any of the above should also be included within the scope of computer readable media.

[0024] The system memory 130 includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) 131 and random access memory (RAM) 132. A basic input/output system 133 (BIOS), containing the basic routines that help to transfer information between elements within computer 110, such as during start-up, is typically stored in ROM 131. RAM 132 typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit 120. By way of example, and not limitation, FIG. 1 illustrates operating system 134, application programs 135, other program modules 136, and program data 137.

[0025] The computer 110 may also include other removable/non-removable, volatile/nonvolatile computer storage media. By way of example only, FIG. 1 illustrates a hard disk drive 140 that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive 151 that reads from or writes to a removable, nonvolatile magnetic disk 152, and an optical disk drive 155 that reads from or writes to a removable, nonvolatile optical disk 156, such as a CD ROM or other optical media. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 141 is typically connected to the system bus 121 through a non-removable memory interface such as interface 140, and magnetic disk drive 151 and optical disk drive 155 are typically connected to the system bus 121 by a removable memory interface, such as interface 150.

[0026] The drives and their associated computer storage media discussed above and illustrated in FIG. 1, provide storage of computer readable instructions, data structures, program modules and other data for the computer 110. In FIG. 1, for example, hard disk drive 141 is illustrated as storing operating system 144, application programs 145, other program modules 146, and program data 147. Note that these components can either be the same as or different from operating system 134, application programs 135, other program modules 136, and program data 137. Operating system 144, application programs 145, other program modules 146, and program data 147 are given different numbers here to illustrate that, at a minimum, they are different copies. A user may enter commands and information into the computer 20 through input devices such as a keyboard 162 and pointing device 161, commonly referred to as a mouse, trackball or touch pad. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). A monitor 191 or other type of display device is also connected to the system bus 121 via an interface, such as a video interface 190. In addition to the monitor, computers may also include other peripheral output devices such as speakers 197 and printer 196, which may be connected through an output peripheral interface 195.

[0027] The computer 110 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 180. The remote computer 180 may be a personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 110, although only a memory storage device 181 has been illustrated in FIG. 1. The logical connections depicted in FIG. 1 include a local area network (LAN) 171 and a wide area network (WAN) 173, but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

[0028] When used in a LAN networking environment, the computer 110 is connected to the LAN 171 through a network interface or adapter 170. When used in a WAN networking environment, the computer 110 typically includes a modem 172 or other means for establishing communications over the WAN 173, such as the Internet. The modem 172, which may be internal

or external, may be connected to the system bus 121 via the user input interface 160, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 110, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, FIG. 1 illustrates remote application programs 185 as residing on memory device 181. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Exemplary XML Data

[0029] The present invention may be used to validate a collection of data or "message" that is organized hierarchically, such as data in the form of eXtensible Markup Language (XML). XML is a meta-language that allows data to be organized according to a defined structure or schema. FIG. 2 shows an example of XML data, where the organization depicted may be useful for representing a post-office address.

[0030] In XML, data items are associated with element names or "tag" names. The tags may be used as delimiters, using the syntactic convention that the beginning of delimited data is marked by the tag name enclosed in angular brackets "<" and ">", and the end of the delimited data is marked by the tag name preceded by a slash and enclosed in angular brackets. The following is an example of XML data that follows this convention:

```
<A>
        Data that is delimited by tag "A".
</A>
```

[0031] In FIG. 2, message 200 is an address. The delimiters <ADDRESS> (202) and </ADDRESS> (222) enclose the entire message, showing that all of the data is part of the "ADDRESS" element. The ADDRESS element comprises two sub-elements: a STREET element and a CITYSTATEZIP element. The STREET element is set to the value "123 Main Street" (204). The CITYSTATEZIP element is a composite element that includes other elements; these included elements are enclosed in delimiters 206 and 220. The CITYSTATEZIP element includes a CITY element that is set to "Redmond" (208), a STATE element that is set to "Washington" (210), and a ZIP element, which is a composite whose included elements are marked by the delimiters 212 and 218. The ZIP element includes the element FIVEDIGIT, which is set to "98052" (214), and the

element PLUSFOUR, which is set to "0123" (216). Thus, message 200 describes an address as a hierarchical collection of data.

[0032] It will be appreciated that an XML message corresponds to a tree data structure. FIG. 3 shows the tree 300 that corresponds to message 200. Node 302 corresponds to the "ADDRESS" tag of FIG. 2; all of the subtrees of node 302 correspond to the elements that are included within the <ADDRESS> and </ADDRESS> delimiters 202 and 222 (shown in FIG. 2). These included elements are the STREET element 204, as well as the CITYSTATEZIP element. As described above, CITYSTATEZIP is a composite that includes other elements. Node 304 corresponds to the CITYSTATEZIP tag, and all of the subtrees of node 304 correspond to the elements that are included within the <CITYSTATEZIP> and </CITYSTATEZIP> delimiters 206 and 220 (shown in FIG. 2). These included elements are the CITY element 208, the STATE element 210, and the ZIP element. The ZIP element, in turn, is a composite that includes other elements. Thus, node 306 corresponds to the ZIP tag, and all of the subtrees of node 306 correspond to the elements that are included within the <ZIP> and </ZIP> delimiters 212 and 218. These included elements are the FIVEDIGIT element 214 and the PLUSFOUR element 216. As can be seen, tree 300 corresponds to the structure in XML message 200, and shows the hierarchical structure of a post-office address contained in message 200.

[0033] Messages generally obey some type of predefined structure or "schema." For example, a schema for "address" messages may define the various different types of structures that may be present in an address. For example, such a schema may define an address message as comprising an ADDRESS tag, and may further define the details of the permissible structure of an ADDRESS tag. For example, the schema may state that an ADDRESS tag contains a STREET element, and also one other element which may be CITYSTATEZIP (for United States addresses), or CITYPROVINCEPOSTCODE (for Canadian addresses), or CITYCOUNTRY for addresses outside of the United States or Canada. For each type of element (e.g., CITYSTATEZIP), the schema may define the permissible structure of that element's constituent sub-elements. Under this definition of an ADDRESS tag, it will be understood that the message 200 and its corresponding tree 300, as shown in FIGS. 2 and 3, are merely one example of permissible content for an ADDRESS tag – i.e., the structure for a United States address – and a legitimate ADDRESS could, alternatively, contain one of the other types of elements mentioned above.

[0034] In effect, the schema defines the permissible syntax for an XML message. However, it is possible to form an XML message that is syntactically correct under some schema, but violates some rule of substance. The invention provides a mechanism that may be used to validate the substance of an XML message, as more particularly described below.

Tree Traversal

[0035] As described above in connection with FIG. 3, an XML message can be regarded as a tree. In accordance with one feature of the invention, the tree represented by an XML message is traversed, preferably in depth-first order. As each node in the tree is encountered, the node is validated in accordance with information provided by one or more validation tables. What it means to "validate" a node – and the structure of the validation tables themselves – is more particularly described below. However, with reference to FIG. 4, an example is described showing what it means to traverse a tree in depth-first order.

[0036] Tree 400 comprises a plurality of nodes. The root of tree 400 is labeled "A". The children of the root node are labeled "A1", "A2", and "A3". Each of these children, in turn, has children, which are labeled "A11", "A12", "A21", "A22", "A31", and "A32", as shown in the figure.

[0037] When tree 400 is traversed in a depth-first traversal order, each node of the tree is encountered in a particular sequence. The algorithm to traverse a tree in the depth-first sequence can be defined recursively. For a given tree:

- First visit the root node of the tree;
- Then, visit each of the subtrees of the root node, if any, in left-to-right order, in depth-first sequence.

Since each subtree is, in effect, a tree with its own root node (e.g., A1 is the root node of one of A's subtrees), the algorithm to traverse a subtree in depth-first sequence is the same as the algorithm to traverse the top-level tree in depth-first sequence. Thus, as will be appreciated by those of skill in the art, this traversal algorithm can be implemented as a recursive function.

[0038] Thus, when tree 400 is traversed using this recursive algorithm, the first node encountered is root node "A". Then the left-most subtree of root node "A" is traversed in depth-first order. Since each subtree is, in itself, a tree, each of these subtrees has a root node. The root node of A's left-most subtree is labeled "A1." Thus, after node A is visited, the tree-traversal algorithm is

applied to the subtree rooted at A1, and the first node visited in this subtree is A1. A1 also has subtrees, and these sub-trees are visited in left-to-right order. Thus, after A1 is visited, the subtree whose root is A11 is traversed in depth-first order. Since A11 has no children, there are no subtrees of A11 to traverse, so the next subtree to the right of A11 (i.e., the tree rooted at A12) is traversed in depth-first order. Since A12 also has no children, there are, again, no subtrees to traverse. Thus, the traversal of A1 and its subtrees is complete, and the algorithm moves on to traverse the next subtree of node A (i.e., the subtree rooted at A2), and proceeds to traverse this subtree in the same manner as the subtree rooted at A1 was traversed.

[0039] When a tree is traversed using the recursive algorithm described above, the result is to visit all of the nodes in tree 400 in the order shown in FIG. 4 – i.e., in the order: A, A1, A11, A12, A2, A21, A22, A3, A31, A32.

Example Validation Table

[0040] As noted above, an XML message may conform to a structure or syntax specified in a schema, and yet still may contain invalid data in its substance. For example, as described above in connection with FIGS. 2 and 3, a message may contain an address, which may specify city = "Redmond," state = "Washington," and zip = "98052." This information is syntactically valid (i.e., including a city, state, and zip code is the correct syntax for an address in the United States), and it is also substantively valid (i.e., 98052 is, in fact, the correct zip code for Redmond, Washington). It should be noted, however, that there is no way to determine from the syntax of an address whether the address is substantively valid. For example, "Redmond, Washington 19103" is a syntactically valid address, but not a substantively valid address, since 19103 is the zip code for Philadelphia, Pennsylvania, not Redmond, Washington. Thus, ensuring that an XML message conforms to a schema can ensure only syntactic validity, but not other types of validity. The invention provides a framework for performing arbitrary validity tests on an XML message.

[0041] FIG. 5 shows an exemplary validation table 500, which specifies the validation tests that are to be performed on a particular type of XML message. The elements shown in table 500 correspond to the element types of tree 400 (shown in FIG. 4) – i.e., tree 400 has elements named A, A1, A11, etc., and so does table 500. Validation table 500 comprises four columns 502, 504, 506, and 508. For a given row of table 500, the entries have the following meanings:

- The entry in column 502 is the name of the element to which the row applies;

- The entry in column 504 indicates whether the delegate(s) specified for the element are "exclusive." (The terms "delegate" and "exclusive" are more particularly explained below.);

- The entry in column 506 is the name for the delegate (or "handler") that will be invoked when the element is encountered; and

- The entry in column 508 is the delegate that will be invoked after the element, and all of its subtrees have been processed.

Thus, the first row in table 500 indicates that, for elements named "A", the delegate named "fA" is to be invoked when the element is encountered, and the delegate named "postfA" is to be invoked after all of the subtrees for the element have been validated. Additionally, this row indicates that the delegates specified in this row are not "exclusive" – i.e., they are not to exclude the use of subtree delegates.

[0042] A "delegate" is a named piece of code that performs the validation for a particular type of element. For example, a delegate for the CITYSTATEZIP element discussed in FIG. 2 could contain (or, at least, consult) tables of United States cities, states, and zip codes to determine whether the combination specified in a CITYSTATEZIP element is a correct combination. It should be understood that the invention is not limited to any particular type of validation test, and consumers of systems that incorporate the invention are free to write their own delegates and define what it means for an element to be valid. In particular, it should be observed that a delegate is not limited to considering the content of the element for which it is registered (or even of the subtrees for which it is registered), but can also consider information outside of that subtree (e.g., a delegate for the CITYSTATEZIP element could also consider the street address (element 204, shown in FIGS. 2 and 3) in order to determine whether the zip+4 value is valid). The fact that validity can be arbitrarily defined based on what delegates are written provides the flexibility to validate any type of message.

[0043] In one example, the delegate is written in a programming language such as C#, although it will be understood that a system according to the invention can be configured to use any named piece of code as a delegate.

[0044] Delegate names appear in both column 506 and column 508. The delegate named in column 506 is the delegate that will be invoked when a given element is visited in a tree traversal. The delegate named in column 508 will be invoked after all of the subtrees of the element have been

traversed. Thus, the delegate named in column 508 is a "post-handler," since it is invoked after the element and all of its subtrees have been visited.

[0045] The "exclusive" property specified in column 504 specifies whether the delegates for subtrees of the current element are to be invoked. Thus, in table 500, element A2 has the exclusive property, meaning that, when an element of type A2 is encountered, the individual delegates for the elements of A2's subtrees are not invoked, because A2's delegates have "exclusive" control over A2 and all of its subtrees. Thus, even though table 500 shows delegates for elements A21 and A22, these delegates are not invoked, because A2 has the "exclusive" property, which prevents delegates from being invoked for any element in any of the subtrees of A2.

[0046] A system in accordance with the invention may contain a validation engine, which walks through an XML message (e.g., in a depth-first order traversal of the tree that the message represents), and applies delegates in accordance with the specification in a validation table. It will be appreciated that the validation engine allows validation tests to be modified or substituted without the need for access to the validation engine's source code; validation tests can be changed simply by registering or de-registering delegates from the validation table. This ability to modify validation tests without access to the underlying validation engine source code is an example of the flexibility provided by the invention.

[0047] Table 1 shows an example validation test performed on tree 400 (shown in FIG. 1) by a validation engine, using validation table 500. The column on the left shows various events in the validation process (e.g., the encountering of elements and the completion of an element's subtrees) in the order in which those events occur. The column on the right shows which delegate is called upon the happening of an event. As can be seen in Table 1, even though elements A21 and A22 have delegates registered for them, those delegates are not called by the validation engine because A21 and A22 are children of node A2, and node A2 is registered with the exclusive flag set to "yes." (If, in a different message, A21 or A22 were to appear as children of some other node which had not been marked as exclusive, then the validation engine would encounter nodes A21 or A22, and the delegate fA21orA22 would be invoked at the time those nodes were encountered.

Table 1

Element encountered by engine	Delegate called

A	fA
A1	fA1
A11	fA11
Subtree at A11 is done	(post-handler null, so nothing called)
A12	fA12
Subtree at A12 is done	postfA12
Subtree at A1 is done	(post-handler null, so nothing called)
A2	fA2
Subtree at A2 is done (fA2 is exclusive)	postfA2
A3	(handler null, so nothing called)
A31	fA31
Subtree at A31 is done	(handler null, so nothing called)
A32	(no registration, so nothing called)
Subtree at A32 is done	(no registration, so nothing called)
Subtree at A3 is done	postfA3
Subtree at A is done	postfA

[0048] In accordance with one feature of the invention, there may be both a global validation table and a local validation table. The architecture of the present invention allows for the possibility that fields or group of fields could be common among several different classes of messages. The settings for these common fields can be stored in a global validation table. Any settings that apply only to a particular class of message can be stored in the local validation table. In practice, the local validation table functions as a set of class-specific overrides for the global validation table. Also, in one embodiment of the invention, multiple global validation tables can be created, and a particular global table can be selected for use based on some context.

Exemplary Validation Process

[0049] As noted above, a validation engine in accordance with the invention walks through an XML message, and invokes delegates to the various elements in the message in accordance with one or more validation tables. The following is a description of the process carried out by a validation engine.

[0050] The validation engine starts with the root element of the XML message (e.g., the ADDRESS element in the example of FIGS. 2-3, or the "A" element in the example of FIG. 4). This starting element is set to be the "current" element. Then for the current element (which shall be referred to in this description as "X"), the validation engine:

- Looks up the delegate in the "element handler" column for X in the local validation table. If a delegate is specified in that column in the local validation table, the engine calls the element handler.

- If the local table does not specify a delegate for X in the "element handler" column, the engine looks up the delegate for X in the global table. If a handler for X is specified in the "element handler" column in the global table, then the specified delegate is invoked. If neither table specifies a delegate for the element X, then no delegate for element X is invoked.

- For each of the child elements of element X, the steps described above are repeated (unless X is registered as "exclusive"). In one embodiment, those steps are implemented in a recursive functions, which may be called on the subtrees of element X.

- If a post-handler delegate exists for X (either in the local or global tables), then the post-handler is called after all subtrees of X have been handled. (If a post-handler delegate exists for X in both tables, then only the delegate specified in the local table is invoked.)

[0051] FIG. 6 shows a process carried out by a validation engine in the form of a flow diagram. Initially, the "current node" is defined to be the root node of a tree (602) (e.g., the root element in an XML message). The local validation table is then consulted. If there is an entry for the current node in the local validation table showing a delegate for that node type (604), then the entry from the local validation table is used (606). If there is no entry in the local validation table, then it is determined whether there is an entry in the global validation table (607). If there is an entry from the global validation table, then that entry is used (608). Assuming that some delegate has been found in either the local or global validation table, that delegate is invoked (610); otherwise, the process continues to 614 as described below.

[0052] If the entry for the current node (i.e., the entry from the local or global validation table – whichever table was, in fact, used), indicates that the entry is not "exclusive" (612), then the process of FIG. 6 is recursively called on each of the child nodes of the current nodes (614). Preferably, the process is called on these child nodes in left-to-right order. If the entry for the current node indicates that the entry is exclusive, then the process of FIG. 6 is not called on the child nodes, since the entry for the current node has exclusive control over all subtrees of the current node.

[0053] If there is a post-handler delegate for the current node (616), that post-handler is invoked (618). (The post-handler is applied either directly after the delegate invoked at 610 (in the case where the current node entry is exclusive), or after handling of the subtrees (in the case where the current node entry is not exclusive)). After the post-handler delegate is applied (or after it has been determined that there is no post-handler delegate), it is determined whether the current node is the root of the top-level tree (620). If the current node is the root of the top-level tree (e.g., node 302 in FIG. 3, or node "A" in FIG. 4), then the process terminates. Otherwise, the process returns to its caller (since, if the current node is not the top level root, the process of FIG. 6 has been called recursively by a prior instances of that process).

[0054] It is noted that the foregoing examples have been provided merely for the purpose of explanation and are in no way to be construed as limiting of the present invention. While the invention has been described with reference to various embodiments, it is understood that the words which have been used herein are words of description and illustration, rather than words of limitations. Further, although the invention has been described herein with reference to particular means, materials and embodiments, the invention is not intended to be limited to the particulars disclosed herein; rather, the invention extends to all functionally equivalent structures, methods and uses, such as are within the scope of the appended claims. Those skilled in the art, having the benefit of the teachings of this specification, may effect numerous modifications thereto and changes may be made without departing from the scope and spirit of the invention in its aspects.